

WEST

Generate Collection

Print

L8: Entry 356 of 467

File: USPT

Apr 14, 1998

DOCUMENT-IDENTIFIER: US 5740413 A

TITLE: Method and apparatus for providing address breakpoints, branch breakpoints, and single steppingAbstract Text (1):

A method and apparatus for providing address breakpoints, branch breakpoints, and single stepping is described. According to one aspect of the invention, a processor is provided which generally includes an execution unit, a first storage area, and an address breakpoint unit. The execution unit recognizes a first debug event in response to the execution of an instruction which causes a branch to be taken. The first storage area has stored therein information. The address breakpoint unit is coupled to the first storage area to receive the information. The address breakpoint unit is also coupled to the execution unit to receive addresses. The address breakpoint unit determines whether the addresses it receives form the execution unit are identified by the information. The execution unit recognizes a second debug event when the address breakpoint unit indicates one of these addresses is identified by the information.

Application Filing Date (1):19970723Brief Summary Text (2):

Ser. No. 08/438,473, titled "A Method and Apparatus for Providing Breakpoints on a Selectable Address Range," filed May 10, 1995, by Gary N. Hammond and Donald Alpert.

Brief Summary Text (3):

Ser. No. 08/454,087, titled "A Method and Apparatus for Providing Breakpoints on Jumps and Software Profiling in a Computer System," filed May 30, 1995, by Gary N. Hammond and Donald Alpert, U.S. Pat. No. 5,659,679.

Brief Summary Text (9):

Software debugging is the detecting, locating, and correcting of logical and/or syntactical errors in a computer program. Debug hardware is often included in a processor for use by a software debug program. Typically, the debug program uses the debug hardware to allow the programmer to examine data and check conditions during the execution of another computer program. Thus, the debugging features of a processor provide programmers with valuable tools for looking at the dynamic state of the processor.

Brief Summary Text (11):

One processor which includes debugging hardware is the Intel.RTM. 80960KB manufactured by Intel Corporation of Santa Clara, Calif. The 80960KB provides several debugging modes, including a "branch -trace mode" and an "instruction -trace mode." When the instruction -trace mode is enabled, the processor generates an instruction -trace event each time an instruction is executed. Debug software can use the instruction -trace mode to "single-step" the processor--i.e., interrupt execution after each instruction to allow the debug software to perform various debug techniques. When the branch -trace mode is enabled, the processor generates an branch -trace event each time an instruction that causes a branch (also termed as a "jump") to be taken is executed. An instruction that causes a branch to be taken is one that causes the processor to transfer flow of execution to another instruction

(e.g., a jump instruction, a branch instruction, etc.). A taken branch typically transfers the flow of execution in a non-sequential manner--i.e., to an instruction which does not sequentially follow the instruction causing the branch to be taken. A branch-trace event is not generated for conditional-branch instructions that do not result in a branch being taken.

Brief Summary Text (12):

A register on the 80960 processor is used to store " trace controls" (flags for selecting between the different trace modes). This register also stores a trace-enable flag for enabling the trace modes selected by the trace controls. Typically, software selects the traces modes to be used through the trace controls and then sets the trace-enable flag when tracing is to be used. Upon recognizing an event, the processor interrupts execution of the current process, stores the interrupted process' execution environment (including the state of the trace-enable flag), disables the trace modes by clearing the trace-enable flag, and invokes the appropriate operating system handler. Upon completing the servicing of the event, the invoked handler instructs the processor to resume execution of the interrupted process using the previously stored execution environment (including the previous state of the trace-enable flag). Thus, the processor disables tracing during the execution of operating system routines which are called by the processor's event handling mechanism. As a result, the debug modes cannot be used to debug operating system routines as they are invoked by the event handling mechanism of the processor. Another limitation of the 80960KB processor is that it does not support address breakpoints.

Brief Summary Text (13):

Another processor which includes debugging hardware is the Intel Pentium.RTM. processor manufactured by Intel Corporation. The Pentium processor includes several debugging features, including a "single-step trap" and "address breakpoints." When enabled, a single-step trap occurs after the execution of the current instruction. Debug software can use the single-step trap to single step the processor. One limitation of this processor is that the INT instructions clear the TF flag. Therefore, software debuggers which single-step code must include complex algorithms to recognize and emulate INT n or INTO instructions rather than executing them directly. Due to this required emulation, the inclusion of new instructions may require the software debuggers to be rewritten. Another limitation of this processor is that additional circuitry had to be included to reconcile the single-step trap with the other events. This circuitry terminates single stepping if an external interrupt occurs. In addition, when both an external interrupt and a single-step interrupt occur together, this circuitry clears the TF flag, saves the return address or switches tasks, and examines the external interrupt input before the first instruction of the single-step handler executes. If the external interrupt is still pending, then it is serviced--i.e., that external interrupt's handler is executed. During the execution of the external interrupt's handler, the single-step trap is disabled. Upon completion of the external interrupt's handler, the processor returns to executing the single-step handler. Thus, operating system routines (such as the external interrupt handlers) are not normally run in single step mode. To run operating system routines in the single step mode, an INTn instruction which calls an interrupt handler must be single stepped. As a result, this processor does not allow single stepping to be enabled on operating system routines as they interact with other programs.

Brief Summary Text (14):

As previously stated, the Pentium processor also supports address breakpoints. The Pentium processor includes 4 address breakpoint registers, each capable of storing the linear address of a breakpoint (i.e., the address of a memory or I/O location). Each of these breakpoint registers has two corresponding enable bits--a local enable bit and a global enable bit. These enable bits are used to enable and disable recognition of address breakpoints in relation to task switches. Task switching, also commonly referred to as multitasking, is the allocation of processing time between different programs executing on the processor. A task switch is like a procedure call, but it saves more processor state information. The Pentium processor supports both hardware task switching (also termed as protected-mode multitasking) and software task switching. Hardware task switches are performed by special hardware on the Pentium which stores nearly all of the processor's registers (e.g.,

the instruction pointer, status registers, general purpose registers, etc.) in memory. The local enable bits are automatically cleared by the processor with every hardware task switch to avoid unwanted breakpoint conditions in other tasks. In contrast, the global enable bits are not cleared by a hardware task switch. One limitation of hardware task switching is that it requires microcode. Future processors may not include microcode. Another limitation of hardware task switching is that it saves nearly all the processor's registers, whether or not they need to be saved. This unnecessary saving of information degrades the processor's performance.

Brief Summary Text (15):

In contrast, a software task switch is one in which the hardware on the processor stores the minimum information necessary (e.g., the instruction pointer and some status registers) to resume the previous task and the software determines whether it is necessary to store the rest. Software task switches can be used to optimize performance by allowing smaller processor states to be saved. For a further description of the Pentium processor see: Pentium Processor's Users Manual--Volume 3: Architecture and Programming Manual, 1994, available from Intel Corporation of Santa Clara, Calif. A limitation of the Pentium microprocessor is that neither the local enable bits nor the global enable bits are cleared by software task switches. As a result, recognition of an address breakpoint is either enabled or disabled during execution of both applications and invoked operating system routines when software tasks switches are used. For example, a programmer could not enable recognition of an address breakpoint only during the execution of operating system handlers. To recognize an address breakpoint, comparing circuitry (typically a comparator) is used. Since the Pentium processor is superscalar (i.e., it allows for the execution of multiple instructions per clock cycle, each instruction being executed by a separate pipeline), an address breakpoint could be generated by the execution of any one of the instructions. To avoid the inclusion of comparing circuitry in every pipeline, the Pentium processor operates in a scalar mode (i.e., using only one pipeline--executing only one instruction per clock cycle) while address breakpoints are enabled. Thus, the Pentium processor trades cost and complexity for a degradation in performance when using address breakpoints.

Brief Summary Text (16):

Typically, programmers are debugging either an application or an operating system. However, when address breakpoints are enabled and software task switches are used, recognition of address breakpoint events is enabled and the processor operates in the scalar mode during the execution of both applications and the operating system. As a result, the Pentium processor is unnecessarily degraded when using address breakpoints. This is especially problematic when debugging software on a multi-user system. Another limitation of the Pentium processor is that it does not support branch breakpoints.

Brief Summary Text (18):

A method and apparatus for providing address breakpoints, branch breakpoints, and single stepping is described. According to one aspect of the invention, a processor is provided which generally includes an execution unit, a first storage area, and an address breakpoint unit. The execution unit recognizes a first debug event in response to the execution of an instruction which causes a branch to be taken. The first storage area has stored therein information. The address breakpoint unit is coupled to the first storage area to receive the information. The address breakpoint unit is also coupled to the execution unit to receive addresses. The address breakpoint unit determines whether the addresses it receives from the execution unit are identified by the information. The execution unit recognizes a second debug event when the address breakpoint unit indicates one of these addresses is identified by the information.

Detailed Description Text (11):

Branch breakpoint unit 190 is included in execution unit 142. Branch breakpoint unit 190 includes circuitry for detecting whether an instruction currently being executed by the processor is causing a branch (also termed as a "jump") to be taken. An instruction that causes a branch to be taken is one that causes the processor to transfer flow of execution to another instruction (e.g., a jump instruction, a branch instruction, etc.). A taken branch typically transfers the flow of execution

in a non-sequential manner--i.e., to an instruction which does not sequentially follow the instruction causing the branch to be taken. A branch breakpoint event is not generated for conditional-branch instructions that do not result in a branch being taken. While enable bit 152 indicates the enable state, branch breakpoint unit 190 transmits a signal each time it detects a branch is or will be taken. Upon receiving this signal from branch breakpoint unit 190, execution unit 142 recognizes a branch breakpoint event (also termed as a "break on jump event"). For a further description of an embodiment of branch breakpoint unit 190, see "A Method and Apparatus for Providing Breakpoints on Jumps and Software Profiling in a Computer System," filed on May 30, 1995, Ser. No. 08/454,087, U.S. Pat. No. 5,659,679. Processor 110 may also include a storage area for storing the source address of a taken jump as described in the above referenced document.

Detailed Description Text (12):

Address breakpoint unit 194 is used for allowing address breakpoints. In one embodiment, address breakpoint unit 194 includes a number of address breakpoint registers and a corresponding number of breakpoint mask registers. Each address breakpoint register is used for storing a breakpoint address, while each of the breakpoint mask registers are used for storing a mask. Each breakpoint address and its corresponding breakpoint mask define an address range. Address breakpoint unit 194 also includes circuitry coupled to the address breakpoint registers and the breakpoint mask registers. In response to receiving designated addresses from internal bus 144, this circuitry determines whether these addresses are within any of the address ranges defined by the information in the address breakpoint registers and the breakpoint mask registers. This embodiment of address breakpoint unit 194 includes additional circuitry which is not necessary to understanding the invention. For a further description of this embodiment of address breakpoint unit 194, see "A Method and Apparatus for Providing Breakpoints on a Selectable Address Range," filed on May 10, 1995, Ser. No. 08/438,473. While this embodiment is described in relation to using addresses and masks to define address ranges, alternative embodiments could use any number of techniques. For example, an alternative embodiment could store a starting address and an ending address to define an address range. As another example, an alternative embodiment could store address ranges in a similar fashion to the Intel Pentium microprocessor manufactured by Intel Corporation of Santa Clara, Calif. For a further description of the Pentium processor see: Pentium Processor's Users Manual--Volume 3: Architecture and Programming Manual, 1994, available from Intel Corporation of Santa Clara, Calif.

Detailed Description Text (13):

Debug control register 180 is used for storing control information for branch breakpoint unit 190 and address breakpoint unit 194. In one embodiment, debug control register 180 contains a type indication corresponding to each address range identified by the breakpoint address registers and the breakpoint mask registers. This type indication is used for specifying whether its corresponding address range is an instruction address breakpoint range or a data address breakpoint range. In one embodiment, the type indication comprises several bits and can further identify the following type of breakpoints: 1) an instruction address breakpoint; 2) a data write address breakpoint; 3) an I/O access breakpoint; or 3) a data read or write address breakpoint. Debug control register 180 is shown containing type bits 186 representing the type indications for the address ranges defined by breakpoint address registers and the breakpoint mask registers.

Detailed Description Text (14):

Debug control register 180 also contains an enable indication for each address range identified by the contents of the breakpoint address registers and the breakpoint mask registers. Each enable indication specifies whether its corresponding address range is currently enabled. Debug control register 180 is shown containing enable bits 188 representing the enable indications for the address ranges defined by breakpoint address registers and the breakpoint mask registers.

Detailed Description Text (15):

Upon receiving an address, address breakpoint unit 194 transmits a signal to execution unit 142 each time: 1) the address is identified by one or more of the address ranges stored in the address breakpoint registers and the breakpoint mask registers; 2) at least one of the address ranges within which the address falls is

enabled (the appropriate enable indication(s) in debug control register 180 indicate the enable state--e.g., enable bits 188); 3) at least one of the address ranges within which the address falls is of the same type as the address (the appropriate type indications in debug control register 180 indicate the appropriate type--e.g., type bits 186); and 4) address breakpoint unit 194 is enabled (enable bit 154 indicates the enable state). Upon receiving this signal from address breakpoint unit 194, execution unit 142 recognizes an address breakpoint event.

Detailed Description Text (22):

As shown in step 240, the appropriate handler is executed. Upon completion of the handler, flow passes to step 250. As previously described, in one embodiment the handler for debug events is a generic handler which services multiple events, including address breakpoint events and branch breakpoint events. The debug event handler knows which event has occurred by inspecting an event status register located on processor 110. The event status register stores a number of bits which indicate which event has occurred--e.g., when an event occurs (such as a branch breakpoint event), the state of these bits is altered to indicate which event has occurred. If an address range breakpoint event has occurred, the event status register will indicate which of the address range breakpoints matched. Typically, the last instruction of a handler to be executed instructs the processor to resume the suspended process.

Detailed Description Text (26):

Since address range breakpoints may be set such that they only cover address ranges which are of interest to the programmer, unnecessary address range breakpoints typically will not occur during the execution of code which is not of interest to the programmer. However, branch breakpoints occur whenever a branch is taken and recognition of branch breakpoints is enabled. As a result, if branch breakpoints are enabled and branches are taken in code that is not of interest to the programmer, unnecessary branch breakpoints will be recognized and serviced. The recognition and servicing of these unnecessary branch breakpoints degrades performance of the system. In addition, in one embodiment, processor 110 is a superscalar processor which operates in a scalar mode when recognition of either address breakpoints or branch breakpoints is enabled. As previously described, operating in the scalar mode degrades performance of the processor. Thus, even though recognition of address range breakpoints typically will not occur during the execution of code which is not of interest to the programmer, operating in the scalar mode during the execution of code which is not of interest to the programmer unnecessarily degrades performance. Thus, allowing these debug breakpoints to be disabled when they are not needed improves performance.

Detailed Description Text (27):

For example, an application programmer is often only interested in debugging a particular application. As a result, branch breakpoints and address breakpoints are not of interest to the application programmer during the execution of handlers. Thus, an application programmer can improve performance during the debugging of an application by causing the operating system to alter the states of the enable bits such that recognition of debug events is enabled during the execution of applications and disabled during the execution of handlers. To accomplish this, the application programmer would run a debug program which causes the processor to: 1) switch to the kernel mode; 2) store enable bit 162 and/or enable bit 164 in the enable state; 3) store enable bit 182 and enable bit 184 in the disable state; and 4) switch back to the user mode. By storing enable bit 162 and/or enable bit 164 in the enable state and causing the processor to switch back to the user mode, the states of enable bit 152 and enable bit 154 are selected for the user mode (see step 250 of FIG. 2). The programmer would then instruct the processor to execute an application program to be debugged. Since enable bit 152 and/or enable bit 154 are stored in the enable state, the selected debug event(s) are recognized during the execution of the application in the user mode. However, since enable bit 182 and enable bit 184 are stored in the disable state, debug events are not recognized during the execution of handlers in the kernel mode. In this manner, the performance of debugging applications is improved by avoiding the unnecessary recognition and servicing of branch breakpoint events during the execution of handlers, such as the debug handler. In addition, performance is improved by allowing the processor to execute the handlers in the superscalar mode.

Detailed Description Text (30):

While this embodiment is described in relation to branch breakpoint events and address breakpoint events, storing a bit for identifying whether certain circuitry should be enabled during the servicing of an event could be implemented for any number of events and any number of different circuits. In addition, while this embodiment describes a computer system in which the enablement of the debug events depends on the current state of the enable bits in status register 150 and these enable bits are altered upon entering and exiting the kernel mode, alternative embodiments can use any number of techniques to provide separate indications for separately selecting the enablement of debug events in different modes. For example, an alternative embodiment can have one register in which all of the enable bits are stored and could have circuitry which chooses which bits in that register are currently used based on the current mode of the processor. Furthermore, while the flow diagram in FIG. 2 describes the situation where a process executing in the user mode is interrupted to execute a handler in the kernel mode, handlers could also be interrupted by the recognition of another event (referred to herein as a secondary handler) to execute a another handler (referred to herein as a secondary handler). In such a case, the processor typically remains in the kernel mode until it has completed the execution of both handlers and returns to executing the suspended process.

Detailed Description Text (35):

Thus, the address breakpoint is used to cause a debug event upon the execution of each sequential instruction, while the branch breakpoints are used to cause a debug event when flow of execution branches. In this manner, single stepping is provided using branch breakpoints and address breakpoints. By providing both branch breakpoints and address breakpoints the invention provides for greater flexibility over prior art processors. For example, the programmer can use branch breakpoints, address breakpoint, or both branch breakpoints and address breakpoints. Furthermore, by using the method described herein, the invention is capable of providing for single stepping without the complex circuitry required by prior art processors which provided a separate single stepping mode.

Related Application Filing Date (1):19950619

CLAIMS:

2. The method of claim 1, said step of storing information further including the steps of:

storing a first indication in a first state which enables recognition of address breakpoint events; and

storing a data identifying an address of said second instruction.

3. In a computer system having a processor, said processor operable in a first mode and a second mode, said second mode providing access to storage areas and execution of instructions which are not available in said first mode, a method for single stepping, said method comprising:

A) enabling recognition of branch breakpoints while said processor is operating in said first mode;

B) storing for an address breakpoint event information identifying a second instruction that sequentially follows a first instruction;

C) enabling recognition of said address breakpoint event while said processor is operating in said first mode;

C) executing said first instruction in said first mode;

D) if execution of said first instruction causes flow of execution to branch to a third instruction, causing a branch breakpoint event;

E) if flow of execution passes to said second instruction, causing said address breakpoint event; and

F) in response to either said branch breakpoint event or said address breakpoint event, storing information to cause another address breakpoint to occur on a subsequent instruction to be executed.

4. The method of claim 3, the step of enabling recognition of branch breakpoints further includes the steps of

A1) storing a first indication indicating branch breakpoints are enabled while said processor is operating in said first mode; and

A2) storing a second indication indicating whether branch breakpoints are enabled while said processor is operating in said second mode.

8. The method of claim 3, further comprising the step of:

disabling recognition of branch breakpoints and address breakpoints while said processor is operating in said second mode.

10. The method of claim 9, wherein:

the address breakpoint unit causes a breakpoint event upon detection of an instruction at the target address.

11. A method for performing single stepping using branch and address breakpoints, said method comprising the steps of:

enabling branch and address breakpoints;

recognizing either a branch breakpoint event in response to a first instruction causing a branch or a first address breakpoint event in response to the first instruction not causing the branch;

in response to recognizing either the branch breakpoint event or the first address breakpoint event, storing data to cause a second address breakpoint upon execution of the next instruction to be executed.

15. A method for single-stepping a set of instructions in a program, said method comprising the steps of:

enabling address breakpoints for said set of instructions;

enabling branch breakpoints; and

single-stepping said set of instructions by causing a plurality of branch breakpoint and address breakpoint events, wherein one of a branch breakpoint event and an address breakpoint event is generated in response to each executed instruction of said set of instructions.